

## CONTROL SYSTEM USING PLURAL OBJECTS, AND ITS BUILDING METHOD AND PERIPHERAL DEVICE CONTROL SYSTEM

Patent Number: JP9073395  
Publication date: 1997-03-18  
Inventor(s): KIMURA YOSHIHIRO; HISAMATSU YUTAKA  
Applicant(s):: SEIKO EPSON CORP  
Requested Patent: ☐ JP9073395  
Application Number: JP19950229545 19950906  
Priority Number(s):  
IPC Classification: G06F9/44 ; G06F9/06 ; G06F13/10  
EC Classification:  
Equivalents:

### Abstract

**PROBLEM TO BE SOLVED:** To provide a control system which can be customized more flexibly and easily as a control system consisting of a plurality of common objects and its building method.

**SOLUTION:** When a 1st OCX 10 as an instance of a common object generates and controls another 2nd OCX 11, an interface object 15 which makes possible a bilateral communication is generated and utilized. By this interface object 15, an event generated at the 2nd OCX 11 can be sent back to the 1st OCX 10 and the 2nd OCX 11 can be operated completely from the 1st OCX 10.

Data supplied from the esp@cenet database - I2

THIS PAGE BLANK (USPTO)

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平9-73395

(43) 公開日 平成9年(1997)3月18日

(51) Int.Cl. <sup>6</sup>	識別記号	庁内整理番号	F I	技術表示箇所
G 0 6 F 9/44	5 3 0		G 0 6 F 9/44	5 3 0 A
9/06	5 3 0		9/06	5 3 0 T
13/10	3 2 0		13/10	3 2 0 A C8-12

審査請求 未請求 請求項の数12 O L (全 17 頁)

(21) 出願番号 特願平7-229545

(22) 出願日 平成7年(1995)9月6日

(71) 出願人 000002369

セイコーエプソン株式会社

東京都新宿区西新宿2丁目4番1号

(72) 発明者 木村 芳弘

長野県諏訪市大和3丁目3番5号 セイコーエプソン株式会社内

(72) 発明者 久松 豊

長野県諏訪市大和3丁目3番5号 セイコーエプソン株式会社内

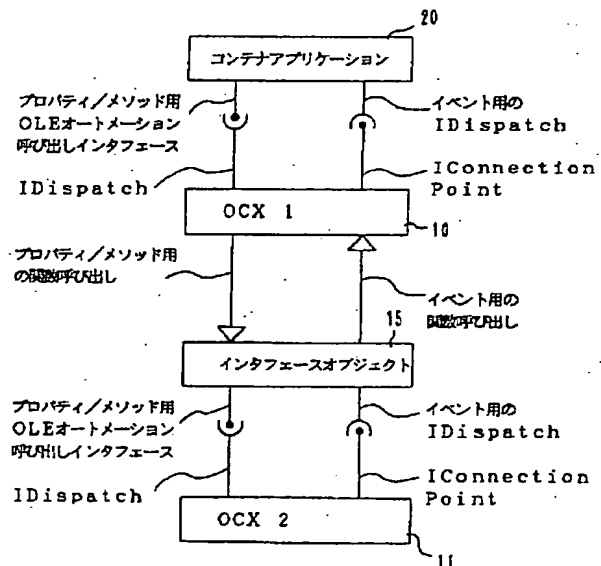
(74) 代理人 弁理士 鈴木 喜三郎 (外1名)

(54) 【発明の名称】 複数のオブジェクトを用いた制御システム、その構築方法および周辺装置制御システム

(57) 【要約】

【課題】 複数のコモンオブジェクトから構成される制御システムにおいて、よりフレキシブルに、また、容易にカスタマイズ可能な制御システムおよびその構築方法を提供する。

【解決手段】 コモンオブジェクトのインスタンスである第1のOCX10が、他の第2のOCX11を作成し制御する際に、双方向の通信が可能なインタフェースオブジェクト15を作成し利用可能とする。このインタフェースオブジェクト15によって、第2のOCX11において発生したイベントを第1のOCX10に返すことが可能となり、第1のOCX10から第2のOCXを完全に動作させることが可能となる。



## 【特許請求の範囲】

【請求項1】 属性値を含むプロパティおよびインプリメントされたファンクションを呼び出すメソッドの少なくともいずれかを提供する第1の機能と、非同期に発生したアクションを含むイベントを発信する第2の機能とを備えた複数の制御オブジェクトを有し、

第1の前記制御オブジェクトまたはそのインスタンスによって少なくとも1つの第2の前記制御オブジェクトのインスタンスが作成され、前記第1の制御オブジェクトまたはそのインスタンスから前記第2の制御オブジェクトのインスタンスの制御が行われるコンピュータ内のシステムであって、

前記プロパティおよびメソッドの少なくともいずれかを前記第1の制御オブジェクトまたはそのインスタンスと前記第2の制御オブジェクトのインスタンスの間で伝達する機能と、

前記第2の制御オブジェクトのインスタンスから前記第1の制御オブジェクトまたはそのインスタンスに前記イベントを伝える機能とを備えたインタフェースオブジェクトまたはそのインスタンスを有することを特徴とする複数のオブジェクトを用いた制御システム。

【請求項2】 請求項1において、複数の前記第2の制御オブジェクトのインスタンスを有し、それら第2の制御オブジェクトのインスタンス毎に前記インタフェースオブジェクトまたはそのインスタンスが作成され、それぞれの前記インタフェースオブジェクトは前記第1の制御オブジェクトまたはそのインスタンスによって参照可能な独自の識別手段を備えていることを特徴とする複数のオブジェクトを用いた制御システム。

【請求項3】 請求項1において、前記第2の制御オブジェクトは複数の前記イベントを第2の配列に従って発信し、前記インタフェースオブジェクトは前記複数のイベントを第1の配列に変換する手段を備えていることを特徴とする複数のオブジェクトを用いた制御システム。

【請求項4】 請求項3において、前記第2の制御オブジェクトのインスタンス毎に前記インタフェースオブジェクトまたはそのインスタンスが作成され、それぞれの前記インタフェースオブジェクトまたはそのインスタンスは前記第1の配列に従って起動される複数のイベント処理手段を備えていることを特徴とする複数のオブジェクトを用いた制御システム。

【請求項5】 属性値を含むプロパティおよびインプリメントされたファンクションを呼び出すメソッドの少なくともいずれかを提供する第1の機能と、非同期に発生するアクションを含むイベントを発信する第2の機能とを備えた複数の制御オブジェクトを有し、第1の前記制御オブジェクトまたはそのインスタンスによって少なくとも1つの第2の前記制御オブジェクトのインスタンスが作成され、これら複数の制御オブジェクトおよびそれらのインスタンスによってコンピュータ内に制御システ

ムを構築する方法であって、

前記プロパティおよびメソッドの少なくともいずれかを前記第1の制御オブジェクトまたはそのインスタンスと前記第2の制御オブジェクトのインスタンスとの間で伝達する機能と、前記第2の制御オブジェクトのインスタンスから前記第1の制御オブジェクトまたはそのインスタンスに前記イベントを伝える機能とを備えたインタフェースオブジェクトまたはそのインスタンスを前記第2の制御オブジェクトのインスタンスに対応して作成することを特徴とする制御システムの構築方法。

【請求項6】 請求項5において、それぞれの前記インタフェースオブジェクトに独自の識別手段が付与され、その識別手段が前記イベントと共に前記第1の制御オブジェクトまたはそのインスタンスに伝えられることを特徴とする制御システムの構築方法。

【請求項7】 請求項5において、前記第2の制御オブジェクトは複数の前記イベントを第2の配列に従って発信し、前記インタフェースオブジェクトは第1の配列に従って起動される複数のイベント処理手段を備えており、前記インタフェースオブジェクトまたはそのインスタンスを作成する際に前記第2の配列を第1の配列に変換する第3の配列を作成することを特徴とする制御システムの構築方法。

【請求項8】 アプリケーションシステムに対し周辺装置の管理を行う制御システムであって、

属性値を含むプロパティおよびインプリメントされたファンクションを呼び出すメソッドの少なくともいずれかを提供する第1の機能および非同期に発生するアクションを含むイベントを発信する第2の機能を備えた複数の制御オブジェクトまたはそのインスタンスを有し、これら複数の制御オブジェクトまたはそのインスタンスからなる制御オブジェクト群は、第1の前記制御オブジェクトまたはそのインスタンスと、

これら第1の制御オブジェクトまたはそのインスタンスによって作成された第2の前記制御オブジェクトのインスタンスと、

前記プロパティおよびメソッドの少なくともいずれかを前記第1の制御オブジェクトまたはそのインスタンスと前記第2の制御オブジェクトのインスタンスの間で伝達する機能、および前記第2の制御オブジェクトのインスタンスから前記第1の制御オブジェクトまたはそのインスタンスに前記イベントを伝える機能を備えたインタフェースオブジェクトまたはそのインスタンスとを備えていることを特徴とする周辺装置の制御システム。

【請求項9】 請求項8において、前記第2の制御オブジェクトは、前記周辺装置の第1の周辺装置に対応していることを特徴とする周辺装置の制御システム。

【請求項10】 請求項9において、前記第2の制御オブジェクトのプロパティは前記第1の周辺装置に固有な仕様に対応した属性値を含み、前記メソッドは前記第1

10

20

30

40

50

の周辺装置に固有な命令を含み、さらに、前記イベントは前記第1の周辺装置において非同期に発生するアクションを反映可能であることを特徴とする周辺装置の制御システム。

【請求項11】 請求項9において、前記第1の制御オブジェクトは、前記インタフェースオブジェクトを介して伝達された前記第2の制御オブジェクトのプロパティ、メソッドおよびイベントの少なくともいずれかを普遍的な仕様に変換することを特徴とする周辺装置の制御システム。

【請求項12】 請求項9において、前記アプリケーションシステムはPOS制御システムであり、前記第2の制御オブジェクトはPOSを構成可能な各種の外部周辺装置毎に用意されており、前記POS制御システムは前記第1の制御オブジェクトまたはそのインスタンスを介して前記外部周辺装置を制御することを特徴とする周辺装置の制御システム。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、共通に使用できる複数のオブジェクトを備え、これらのオブジェクトを用いてカスタマイズ可能な制御システム、その構築方法および周辺装置制御システムに関するものである。

【0002】

【従来の技術】近年、コンピュータを用いて多種多様の処理が行われ、コンピュータに接続可能な外部入出力機器も多種多様なものが用意されている。従って、コンピュータを中心として、ユーザーの環境や目的などに合わせてフレキシブルなシステムを構築し、カスタマイズ可能なシステムを提供することが重要となっている。例えば、共通な規格のバスを用いて1つのコンピュータに様々なタイプのプリンター、ディスプレイ、バーコードリーダーを始めその他の入出力機器が接続可能であり、コンピュータにはこれらの入出力機器を制御可能なソフトウェアがインストールされる場合には、多くの機器を的確に制御するように制御システムが構築されていなければならない。また、複数のコンピュータをネットワークで結んだ環境が簡単に構築できるので、個々のコンピュータによって処理すべき対象およびその手順を様々に変更できることが望ましい。

【0003】このようなフレキシブルなシステムをコンピュータ内に構築する1つの方法として、共通に用いることのできる複数のオブジェクト（コモンオブジェクト）を作成し、これらを活用してオペレーティングシステムやアプリケーションプログラムを構築することが考えられる。マイクロソフト社のOLE（Object Linking and Embedding）およびOLEプログラミング用に用意されたインタフェースを用いたOLEオートメーションあるいはOLEカスタムコントロール（OCX）といったシステムはその1つである。

【0004】

【発明が解決しようとする課題】本発明においては、コモンオブジェクトを活用して多種多様なオペレーティングシステムあるいはアプリケーションプログラムを簡単に構築できるシステムおよびその方法を提供することを目的としている。特に、1つのコモンオブジェクトが他のコモンオブジェクトの機能を十分に活用できるシステムおよびその構築方法を提供することを目的としている。さらに、1つのコモンオブジェクトがクライアントあるいはコントロールとなり、他のコモンオブジェクトをサーバーとして活用する際に、サーバー側で発生したイベントを個別に識別でき、また、その処理手順を記述しやすいシステムおよび構築方法を提供することを目的としている。さらに、多種多様な周辺装置の制御に適したオープンで処理速度の早い制御システムを複数のコモンオブジェクトによって構成可能とすることも本発明の目的の1つである。

【0005】

【課題を解決するための手段】本発明は、コモンオブジェクトとして用いられるように属性値を含むプロパティおよびインプリメントされたファンクションを呼び出すメソッドの少なくともいずれかをアプリケーションプログラム、他の制御オブジェクトあるいはそのインスタンスに提供する第1の機能と、非同期に発生するアクションを含むイベントを発信する第2の機能とを備えた複数の制御オブジェクトを有するシステムを対象としている。このシステムにおいてコンピュータ内に実行可能な制御システムを構築する際に、第1の制御オブジェクトまたはそのインスタンスによって少なくとも1つの第2の制御オブジェクトのインスタンスが作成され、第1の制御オブジェクトまたはそのインスタンスから第2の制御オブジェクトのインスタンスの制御を行うことが考えられる。本発明のこのような制御システムにおいて、プロパティおよびメソッドの少なくともいずれかを第1の制御オブジェクトまたはそのインスタンスと第2の制御オブジェクトのインスタンスの間で伝える機能と、第2の制御オブジェクトのインスタンスから第1の制御オブジェクトまたはそのインスタンスにイベントを伝える機能とを備えたインタフェースオブジェクトを設けるようにしている。従って、本発明により、1つの制御オブジェクトまたはそのインスタンスから、他の制御オブジェクトまたはそのインスタンスをプロパティ、メソッドおよびイベントの全てを含んで双方向の通信が可能なインタフェースオブジェクトおよびそのインスタンスによって連絡することが可能となる。このため、制御オブジェクトとして提供された機能を十分に活用した制御システムを簡単に構築できる。本発明の制御システムにおいては、第1の制御オブジェクトまたはそのインスタンスが、第2の制御オブジェクトまたはそのインスタンスにおいて発生したイベントを受け取ることができ、第1の

制御オブジェクトまたはそのインスタンスがそのイベントに対応した処理を行い、さらに、第1の制御オブジェクトまたはそのインスタンスを活用している他の制御オブジェクトあるいはそのインスタンス、アプリケーションプログラムあるいはオペレーティングシステムに対しイベントを迅速に伝えることが可能となる。従って、本発明により、コモンオブジェクトの機能を完全に活用した制御システムを構築することができる。

【0006】このようなインタフェースオブジェクトまたはそのインスタンスは、第2の制御オブジェクトのインスタンス毎に作成することが望ましく、複数のインスタンスを作成する際は、それぞれのインスタンスに対応するインタフェースオブジェクトに第1の制御オブジェクトまたはそのインスタンスによって参照可能な独自の識別手段、例えば識別番号を設けることが有効である。各インスタンスにおいて発生したイベントは、対応するインタフェースオブジェクトまたはそのインスタンスを介して第1の制御オブジェクトまたはそのインスタンスに伝えられるので、第1の制御オブジェクトまたはそのインスタンスはいずれの作成したインスタンスにおいて発生したイベントであるかを判別できる。

【0007】また、第2の制御オブジェクトが複数のイベントを独自の第2の配列に従って発信してもインタフェースオブジェクトおよび第1の制御オブジェクト側を変更しないで済むように、インタフェースオブジェクトに複数のイベントを第1の配列に変換する手段を設けておくことが望ましい。また、第2の制御オブジェクトにおける配列に係わらず所定の第1の配列に変換されるので、イベント毎に起動されるイベント処理手段の記述も容易となる。

【0008】本発明に係るインタフェースオブジェクトを用いることによって、複数の制御オブジェクトおよびそれらのインスタンスによってコンピュータの機種やそれに接続される周辺装置に対しオープンな制御システムを構築できる。例えば、第2の制御オブジェクトを複数の周辺装置の中の第1の周辺装置に対応して用意することによって、第2の制御オブジェクトに第1の周辺装置に固有な仕様に対応した属性値を含んだプロパティと、命令を含んだメソッドと、さらに、第1の周辺装置において非同期に発生するアクションを反映可能なイベントを設定することができる。そして、第2の制御オブジェクトと第1の制御オブジェクトをインタフェースオブジェクトを介して接続することによって、第1の制御オブジェクトにおいて第2の制御オブジェクトのプロパティ、メソッドおよびイベントの少なくともいずれかを複数の機種、メーカーあるいは複数の種類の周辺装置に対して普遍的に規定された仕様に変換し、周辺装置に依存しない共通のアプリケーションプログラムインタフェース (API) をアプリケーションプログラムや上位の制御オブジェクトおよびそのインスタンスに対し提供す

ることができる。さらに、第1の制御オブジェクトのAPIをコンピュータの機種に依存しない普遍的なものにすることも可能であり、機種依存性がなく、汎用性の高いアプリケーションプログラムとすることができる。従って、本発明の制御システムにより、ユーザーは自己の環境に合わせたアプリケーションおよびハードウェアによってPOSシステムなどの周辺装置を多く使用するシステムを簡単に、そしてフレキシブルに構築できる。

【0009】

【発明の実施の形態】

【OLEオートメーションをベースにした例】以下において、マイクロソフト社の提供するプログラム開発環境であるMicrosoft Foundation Class (MFC) 環境下において本発明を実施した例に基づきさらに詳しく説明する。

【0010】MFCはOLEのプログラミングを容易に行えるようにさまざまなライブラリーを提供しており、オブジェクト指向のプログラミング言語であるMicrosoft社が提供するVisual C++ (以下VC++) などによってOLEオートメーション機能を用いたシステムの開発を行うことができる。本例では、図1に示すように、アプリケーション (コンテナアプリ) 20と、システム内に共通に活用できるように用意されたコモンオブジェクトあるいはそのインスタンス (OCX) 10および11とを階層的に結合して構築された制御システムについて説明する。

【0011】コモンオブジェクトは、他の実行型のソフトウェアを構成するオブジェクトとして提供されても良く、あるいはダイナミックリンクライブラリ (DLL) として提供されるオブジェクトであっても良い。コモンオブジェクトを提供するこれらの実行型のソフトウェアやDLLはEXEサーバーあるいはDLLサーバーと呼ぶことができ、これらに対し、コンテナアプリ20をクライアントあるいはコントロールと呼ぶことができる。OLEに対応したこれらのコモンオブジェクトは、例えば、MFCが提供しているCCmdTargetクラスから派生させることができる。CCmdTargetクラスは、コモンオブジェクトを管理するためのIUnknownインタフェースや、その他のコモンオブジェクトとして必要なインタフェース等をサポートするクラスである。CCmdTargetクラスはコントローラ側からデータを保持した構造体などを受渡しするためのIDispatchインタフェースもサポートしており、このメソッドInvokeをコールすることによってコモンオブジェクト側のプロパティあるいはメソッドを用いることができる。

【0012】プロパティは、そのコモンオブジェクトの有する属性であり、カラー、テキスト、番号、フォントあるいはプッシュボタンがプッシュされた時の動作などが含まれる。メソッドは、コモンオブジェクトにインプ

リメントされた、例えば編集機能などのファンクションであり、メソッドをコールすることにより共通オブジェクトの機能を操作することができる。共通オブジェクトは、プロパティやメソッドを提供する機能に加え、共通オブジェクトに対する外部からのアクション、例えばマウスボタンのクリックやキー入力、その他の非同期に発生するアクションをイベントとして発信する機能を備えることができる。これは通常、コントロール側がプロパティ等をコールすると同様に、共通オブジェクトにおいてInvoke関数をコールすることによって行われる。

【0013】図1には、コンテナアプリ20がある共通オブジェクトあるいはそのインスタンスを用いている制御システムを示してある。本図に示したシステムでは、コンテナアプリ20が第1のオブジェクトのインスタンス(OCX)10を作成し、さらに、その第1のOCX10が別の第2のオブジェクトのインスタンス(OCX)11を作成して制御している制御システムを示してある。MFCライブラリーには、コンテナアプリやOCX等からInvoke関数をコールする複雑さをカバーするためにCOleDispatchDriverクラスが用意されており、OCX10はこのクラスから派生させたオブジェクトのメンバー関数をコールすることにより、他の制御オブジェクトのインスタンスであるOCX11を作成し、OCX11のメソッド等を活用できる。

【0014】サーバー内の共通オブジェクトを用いる場合、例えば、コンテナアプリの動作しているプロセッサと異なるプロセッサでその共通オブジェクトが動作しているケースも考えられる。このようなケースでは、本例のような制御システムを、コントロール側とサーバー側で通信処理を行うことにより構築することができる。また、1つのプロセッサに活用する共通オブジェクトをコピーして制御システムを構築することも可能である。いずれの場合も、用いられる共通オブジェクトのコード自体には変更は全く加えられず、その共通オブジェクトを活性化する毎にデータやスタックなどの領域が用意され、それを用いて共通オブジェクト(サーバー側)とコントロール側が通信しながら制御システムを構築する。従って、同一の共通オブジェクトをコントロール側で複数用いることも可能であり、この場合、共通オブジェクトが複数コピーされるのではなく、データやスタックなどの領域がそれぞれに用意される。このように共通オブジェクトが他のオブジェクト等によって活用可能な状態になることを本明細書ではインスタンスと称する。

【0015】さて、第1のOCX10がコントロール側となって第2のOCX11を呼び出して使用したい場合、VC++のクラスウィザードを用いてCOleDispatchDriverクラスから派生されたインタフェースオブジェクトを用いることが可能であることは上述した通りである。な

お、以下において、インタフェースオブジェクトとは、そのインスタンスも含めた意味で用いている。VC++2.0のCOleDispatchDriverクラスは、メソッドやプロパティをIDispatchインタフェースを介して第2のOCXに渡す機能をサポートしているが、第2のOCXが発信するイベントをとらえる機能をサポートしていない。従って、コンテナアプリ20の側が第1のOCX10の用いる第2のOCX11を識別しており、第2のOCXのイベントを受信する機能を予め備えている場合は何ら問題はない。しかしながら、コンテナアプリ20の側が第1のOCX10の用いる第2のOCX11を識別していない場合は第2のOCX11で発生したイベントをとらえることができない。

【0016】用意されている共通オブジェクトが予め判明していたり、コンテナアプリに合わせて共通オブジェクトが用意されている場合は、共通オブジェクトがコンテナアプリに対してイベントを渡せるようにシステムを構築したり、あるいは、第2のOCXではコンテナアプリに影響を与えるようなイベントを発生させないなどの処理が可能であろう。しかしながら、種々の外部入出力端末がサポートされるなどコンピュータの環境が様々に変化し得る今後の制御システムを考慮すると、よりフレキシブルな制御システムを構築できることが望ましい。このためには、コンテナアプリ20が共通オブジェクトを活用してカスタマイズが可能なように、共通オブジェクトも共通オブジェクトを用いたカスタマイズが可能な環境をサポートすることが考えられる。従って、本発明では、共通オブジェクトから共通オブジェクト15を提供し、さらにフレキシブルな制御システムを簡単に構築できるようにしている。双方向の通信が可能なインタフェースオブジェクトを用いて、共通オブジェクト間を結べるようになれば、コンテナアプリ20の側は第2のOCX11を認識する必要は全くなり、また、第2のOCX11もコンテナアプリ20に左右されないオブジェクトとして提供することができる。

【0017】本例においては、OCXからOCXを使用するためのディスパッチ処理をイベントも含めて完全にサポートされたインタフェースオブジェクトを実現するために、COcxDispatchDriverクラスを提供している。このCOcxDispatchDriverは、CCmdTargetクラスと、COleDispatchDriverクラスとを多重継承(Multiple Inheritance)して派生したオブジェクトクラスであり、次のようになる。

【0018】class COcxDispatchDriver:public CCmdTarget,public COleDispatchDriver { ... };CCmdTargetクラスは上述したように、サーバーとしての機能を備えたオブジェクトを派生するクラスであり、IDispatchインタフェースのサポートされたクラスである。従って、第2のOCXに対してサーバー側として動作しイベ

ントを受け取るインタフェースをサポートしたインタフェースオブジェクトを実現できる。一方、COleDispatchDriverは、第2のOCXに対してクライアントあるいはコントロール側として動作し、プロパティ/メソッドに対するアクセスを簡単に行なえる機能を備えたオブジェクトクラスである。従って、クラスウィザードにより上記の2つのクラスから派生させたオブジェクトクラスであるCOcxDispatchDriverクラスは、上記の2つの主な機能を備え、これらに加えそれぞれのクラスの機能を全て\*

```
Class COcxDispatchDriver : public CCmdTarget,
                           public COleDispatchDriver
COcxDisptchDriver::COcxDispatchDriver() // 生成
public: // メンバ変数
UINT m __ObjID; // オブジェクトインスタンスのID
               EstablishConnection() でユニークに初期化される
IID m __IIDEvents; // イベント インタフェース ID
UNIT m __nEvents; // イベント数
CDWordArray m __dispID // ユーザが定義したディスパッチマップに対応して
                       ディスパッチIDを変換するための配列
public: // メンバ関数
Bool EstablishConnection(
    REFCLSID clsid,
    COleException* pError=NULL); // オブジェクトインスタンス
                                   を作成し、メソッド、プロパティ、イベントへの
                                   アクセスコネクションを有効にする
void DestroyConnection(); // コネクションを開放し、オブジェクト
                           インスタンスを削除する
```

COcxDispatchDriverクラスでは、汎用的なクラスを実現するためにイベントの処理関数（イベントハンドラ）は含んでいない。従って、COcxDispatchDriverから派生されたオブジェクトに特定のOCXのイベントに対応するイベントハンドラを記述する必要がある。まず、OCXの返すイベントの外部名に合わせてディスパッチマップを記述する。ディスパッチマップの各イベントの順番は任意で良く、後述するようにイベントとのコネクションをセットするときに実際に受信するイベントの順番との対応付けがなされる。ただし、OCXの全部のイベントを完全に記述しておく必要がある。

【0021】ディスパッチマップのエントリは例えば以下のようになり、マクロのパラメータは、順に、ディスパッチドライバクラス名、外部イベント名、イベントハンドラ名、戻り値、パラメータ情報となる。

【0022】DISP \_\_FUNCTION\_\_( \_\_DSoprn, "ControlCompleteEvent", ControlCompleteEvent, VT \_\_EMPTY, VTS \_\_I4 VTS \_\_SCODE VTS \_\_PBSTR VTS \_\_I4)このような外部イベント名に合わせたディスパッチマップを記述することにより、OCXで発生したイベントを直観的に判りやすい形式で受け取ることができる。さらに、イベントハンドラのプロトタイプ宣言は下記のようなので、通常の関数の形式で記述でき、プログラム開発が極めて

\*継承し、サポートされたオブジェクトクラスなので、本発明に係るインタフェースオブジェクトを生成するオブジェクトクラスとして適している。CCmdTargetクラスとCOleDispatchDriverクラスには重複する部分がないため、全く問題なく多重継承が可能である。

【0019】このように派生されたCOcxDispatchDriverクラスの仕様は以下の通りである。

【0020】

容易となる。

【0023】void \_\_DSoprn::ControlCompleteEvent(long ControlID, SCODE Result, BSTR FAR\*, pString, long data);次に、COcxDispatchDriverクラスから派生され、上記のようにインタフェースの準備されたインタフェースオブジェクト15と第2のOCX11とのコネクションをセットするステップを図2および図3に示したフローチャートに基づき説明する。本例のインタフェースオブジェクト15においては、そのメンバ関数COcxDispatchDriver::EstablishConnection()を呼び出すだけで、第2のOCXであるオブジェクトインスタンスが作成され、プロパティおよびメソッドを提供し、イベントを受け取るコネクションがセットされる。

【0024】図2に示すように、EstablishConnectionがコールされると、ステップ31において、clsidによって指定された第2のOCXであるオブジェクトインスタンス11が生成され、動作を開始する。変数clsidは作成するOCXを識別するための128ビットの値が格納されている。ステップ32において第2のOCX11が動作を開始すると、ステップ33においてこのインスタンス11のIDispatch インタフェースを取得し保存する。インタフェースオブジェクト15は、このIDispatchを通じてInvoke関数をコールし、インスタンス11に



11

プロパティまたはメソッドを渡すことができる。ステップ34において、第2のOCX11にプロパティあるいはイベントを渡すコネクションがセットされると、ステップ35において、第2のOCX11のイベントのインタフェースがチェックされる。

【0025】ステップ36において、イベントに関するインタフェースがあるとステップ37において、イベントに関する情報を格納する。第2のOCX11において用意されているイベント名リストはEntryNamesに、ディスパッチIDリストはDispIDsに、また、パラメータ情報リストはParamInfoに格納される。ステップ38においてこれらの処理が通常に終了すると、図3に示したイベントの対応付けを行う工程に移行する。一方、コネクションがセットできない場合はステップ39においてエラー処理を行う。

【0026】図3にイベントの対応付けを行う工程を示してある。まず、ステップ41においてカウンタfcntをゼロクリアする。次にステップ42においてカウンタfcntを確認し、ステップ43において第2のOCX11から得られたイベント名EntryNames(fcnt)がインタフェースオブジェクト15のイベントハンドラに予め用意したマップpDispMapにあるか否かをチェックする。マップpDispMapにある場合は、さらに、ステップ44において、パラメータ情報ParamInfo(fcnt)がイベントハンドラに予め用意したマップpDispMapと一致するかを確認する。確認できた場合は、ステップ45において、第2のOCX11から得られたイベントのディスパッチIDであるDispIDs(fcnt)をイベントハンドラに予め用意したマップpDispMapでのインデックスに変換するための配列m\_dispIDを設定する。ステップ46において、カウンタfcntをインクリメントし、すべてのイベントに対し、これらのステップを繰り返す。予め用意したマップpDispMapとの対応が付くと、ステップ47において、第2のOCX11のイベントに関するコネクションをセットする。このステップにおいては、インタフェースオブジェクト15に用意されたイベントを受け取るためのIDispatchのアドレスを、オブジェクト間の結合点を実現するために用意されているインタフェースであるIConnectionPointに対し渡すことによってイベントに関するコネクションをセットする。これによって、第2のOCX11においてInvoke関数をコールすると、イベントをインタフェースオブジェクト15に渡すことができる。すなわち、インタフェースオブジェクト15が第2のOCX11からイベントを受け取ることができる。

【0027】ステップ48においてコネクションのセットが無事に終了すると、ステップ49においてイベントのディスパッチIDを変換するための配列m\_dispIDを保存する。ステップ50において、上記のステップが全て正常に終了するとインタフェースオブジェクト15と第2のOCX11とのコネクションのセットが終了し、

12

第1のOCX10に対し第2のOCX11のプロパティあるいはメソッドを提供可能となり、さらに、第2のOCX11から第1のOCX10にイベントが伝えられるようになる。実際には、インタフェースオブジェクト15が第2のOCX11からイベントを受け取ったことを第1のOCX10に通知する必要がある。従って、第1のOCX10にインタフェースオブジェクト15がイベントを受け取った場合にコールされるためのpublicなメンバ関数が用意され、インタフェースオブジェクト15はそのメンバ関数をコールする。

【0028】このように、本例のインタフェースオブジェクト15を用いることによりコモンオブジェクトあるいはそのインスタンス同士の間で双方向の通信が可能となる。従って、コモンオブジェクトあるいはそのインスタンスを用いた、いっそうフレキシブルな制御システムを構築することができる。さらに、本例のインタフェースオブジェクトにおいては、イベントに関するコネクションをセットするときに、コモンオブジェクト側で用意されたイベントの配列と、インタフェースオブジェクト側で用意したイベントの配列を確認し、対応付けるようにしている。従って、第1のOCX10に対して第2のOCX11のインタフェースが完全に既知となっていなくとも、イベントの種類と数が合致していればコネクションをセットすることができる。すなわち、プロパティ、メソッドおよびイベントの名称が合致していれば複数のコモンオブジェクト間に双方向のコネクションをセットすることができる。このため、コモンオブジェクトの開発が容易となり、バージョンアップが行われたり、異なったオブジェクトが採用された場合であっても、フレキシブルなシステムをより安全、確実に構築することができる。

【0029】さらに、イベントを受け取ったときに起動されるイベントハンドラも通常の関数の形式で記述することができるので、オブジェクトの開発が極めて容易となり短時間で行えるようになる。

【0030】図4に、第2のOCX11から、インタフェースオブジェクト15に用意されたIDispatchインタフェースのイベントを通知する呼び出し Invoke(EventID,...) がコールされた場合の処理を示してある。IDispatchインタフェースは Invoke() および、それをサポートする基本的な関数 (AddRef(), Release(), QueryInterface(IID&, LPUNKNOWN\*) 等) を提供しており、IDispatchインタフェースが、相手のOCXが発生するイベントの IID (Interface ID) にQueryInterface(...) を介して応答する。Invoke() がコールされると、この時点でパラメータとしてイベントの種類を規定するdispatch ID やイベントのパラメータを保持する構造体などがインタフェースオブジェクト15に渡される。インタフェースオブジェクト15では、ステップ60においてパラメータが正当か否かをチェックする。正当の場合はス

ステップ61において、第2のOCX11から得られたイベントの配列dispIDMemberを用意されたイベントハンドラのマップpDispMapに合わせて、変換用の配列m\_dispIDに基づき変換する。ステップ62において、dispIDMemberに対応するイベントハンドラをpDispMapから取り出す。そして、ステップ63において、イベントハンドラが有効か否かを確認し、有効な場合はステップ64においてイベントに対応するイベントハンドラを呼び出し、イベントハンドラに定義された処理、例えば、第1のOCX10にイベントを発信するなどの処理を行う。

【0031】図5に、第1のOCX10から第2のOCXの複数のインスタンス11aおよび11bが作成され、第1のOCX10がこれらの第2のOCX11aおよび11bを制御するシステムを示してある。すなわち、第1のOCX10は、2つの第2のOCX11aおよび11bに対しクライアントあるいはコントローラ側となっている。このような制御システムにおいて、第1のOCX10とこれらのOCX11aおよび11bのインタフェースは、それぞれの第2のOCX11aおよび11bを作成したインタフェースオブジェクト15aおよび15bをそれぞれ介して行われる。本例においては、COcxDispatchDriverから派生したインタフェースオブジェクトが生成される度に、個々のインタフェースオブジェクトを識別可能なIDが用意される。そして、インタフェースオブジェクトからイベントが第1のOCX10に伝えられるときは、個々のインタフェースオブジェクトに固有のIDが第1のOCX10の側から参照できるようになっている。このIDは、インタフェースオブジェクト作成時にユニークな値が割り当てられるようになっているが、必要であればインタフェースオブジェクトを作成した側、すなわち、本例であれば第1のOCXで適当な値に変更することもできる。

【0032】図5に示したような複数の第2のOCX11aおよび11bが作成された場合、個々の第2のOCXにおいて発生したイベントはインタフェースオブジェクト15aおよび15bをそれぞれ介して第1のOCXに伝えられる。従って、第1のOCXは、各々のインタフェースオブジェクトに付された固有のIDによっていずれの第2のOCXで発生したイベントであるか明確に識別することができる。このように、本例のインタフェースオブジェクトにより、ある共通オブジェクトあるいはそのインスタンス（第1のOCX）から複数のインスタンス（第2のOCX）を作成して活用することが可能である。そして、このようなケースであっても作成した側（第1のOCX）は個々のインスタンス（第2のOCX）を識別し管理しなくとも、どのインスタンス（第2のOCX）でイベントが発生したかを識別することができる。さらに、イベントハンドラが、インスタンス（第2のOCX）を発生させたオブジェクトあるいはインスタンス（第1のOCX）に含まれるのではなく、個

々のインタフェースオブジェクト毎に用意される。従って、作成した側（第1のOCX）は、どのインスタンス（第2のOCX）で、どのようなイベントが、どのようなパラメータを伴って発生したかを確実に把握することができる。

【0033】一方、作成されたインスタンス（第2のOCX）の側も、発生したイベントをそれぞれのインタフェースオブジェクトに単に発信すれば良く、インスタンス自身が最初のインスタンスであるか、あるいは他に同じクラスのインスタンスが存在するの否か等の判断をする必要はない。

【0034】このように、本発明にかかるインタフェースオブジェクトを用いることによって、複数の共通オブジェクトおよびそれらのインスタンスを備えた制御システムをフレキシブルに、そして簡単に構築することができる。それぞれの共通オブジェクトには、活用する相手の共通オブジェクトに関し最小限の情報を用意しておけば良く、また、活用される相手の共通オブジェクトに関する情報も最小限で済む。従って、個々の共通オブジェクトのカプセル化をいっそう進めることができる。その一方で、本発明のインタフェースオブジェクトはイベントも含めた双方向の通信機能を備えており、活用する相手の共通オブジェクトおよびそれらのインスタンスを完全に動作させることができ、それらの備えた機能を最大限に発揮させることができる。

【0035】なお、本例においては、第1および第2のOCXからなる2層の制御システムを用いて説明しているが、3層あるいはそれ以上の階層構造をもった制御システムであっても同様に構築することができる。また、上記のインタフェースオブジェクトを用いてアプリケーションと第1のOCXとの間のインタフェースを実現することも可能である。アプリケーションプログラムに限らず、オペレーションシステムなどであっても同様に構築できることはもちろんである。また、上記においては、マイクロソフト社の提供するMFC環境下において本発明のオブジェクトインタフェースを実現した例を用いて説明しているが、MFCライブラリを用いなくとも、同等の機能を備えたオブジェクトインタフェースを作成することは可能である。さらに、共通して活用できるオブジェクトが用意されたシステム環境であれば、マイクロソフト社のOLEオートメーションでなくとも上記と同様の共通オブジェクトを用いたシステムを構築できる。このような共通オブジェクトを用いたシステムは、1つのプロセッサを備えたコンピュータ単体で実現することもでき、また、ネットワーク環境で接続された複数のプロセッサを有するシステムにおいても構築できることは上述した通りである。

【0036】〔周辺装置の制御システム〕図6に、本発明に係るインタフェースオブジェクトと、複数の共通オブジェクトおよびそのインスタンスによって周辺装置

の制御システムを構築した例を示してある。図6にはパーソナルコンピュータ（パソコン）70を中心に構成されたPOSシステムを示してある。パソコン70にはPOSアプリケーションプログラム71がインストールされており、パソコンのオペレーティングシステム（OS）105の上で動作する。OS105は、キーボードドライバ106やモニタディスプレイドライバ107等を介してパソコンとして通常必要な周辺装置を制御する機能を備えており、アプリケーションプログラム71とキーボードあるいはモニタディスプレイ（不図示）との間のデータ転送はOS105を介して行われる。

【0037】POSシステムには、パソコンに通常用意されるこれらの周辺機器に加えて、客先に金額等を表示するカスタマディスプレイ110、レシート等の印刷を行うレシートプリンタ112、チェックなどの印刷を行うスリッププリンタ113、さらに、金銭を保管するキャッシュドロワ115が必要となり、これらの周辺装置はRS-232Cポート等の拡張用のポートに接続される。例えば、カスタマディスプレイ110がRS-232Cポートに接続され、カスタマディスプレイ110をバススルーしてレシートプリンタ112およびスリッププリンタ113を備えたプリンタ111が接続される。キャッシュドロワ115は、プリンタ111の下部に設置され、プリンタ111の制御機構を介して操作される。これらの周辺装置は多くのメーカーから様々な機種が市販されており、ユーザーは自己の環境に適したものを選択してPOSシステムを構築可能である。しかしながら、メーカーや機種の異なる周辺装置は個々に仕様が異なるので、市販されている全ての周辺装置に適合したアプリケーションプログラムの作成は不可能である。さらに、周辺装置がバージョンアップされると、これに伴い仕様も変更になる。従って、従来は、ユーザーは自己に都合の良い周辺機器によってPOSシステムを構築することが困難なこともあり、さらに、周辺機器がバージョンアップされても新しい機種を構築済のPOSシステムにすぐに適用できるとは限らなかった。

【0038】これに対し、上述したOCXによって制御システムを構築すると、非常にオープンなシステムを形成できる。従って、どのような機種の周辺機器を用いたPOSシステムであっても簡単に構築できる。また、周辺機器のバージョンアップに対しても簡単に対処できる。例えば、図6に示す本例の周辺機器の制御システム72は、3つのレベルのOCXを備えている。第1のレベルのOCXとして、レシートプリンタフォーマット変換用OCX73と、スリッププリンタフォーマット変換用OCX74が用意されている。これらのOCX73および74では、例えば、アプリケーションプログラム71から送られた売上品リストや合計金額等のデータを所定のフォーマットに配置する処理が行われる。所定のフォーマットは、OCX73あるいは74の内部に設定さ

れていても良いし、下位レベルの制御用OCXであるレシートプリンタ制御用OCX75あるいはスリッププリンタ制御用OCX76がプロパティとして有しており、これらの制御用OCXのプロパティを得て変換用OCX73あるいは74がフォーマットを設定しても良い。いずれにしても、アプリケーションプログラム71は、出力表示されるフォーマットに関係なく、出力用のデータを変換用OCX73あるいは74に受け渡せば良い。従って、インタフェースの形式を限定でき、汎用性の高いアプリケーションプログラムとして提供することができる。また、アプリケーションプログラムから固有のフォーマットで出力されたデータを、第1レベルのOCXによって下位のOCXに共通するフォーマットに変換することも可能である。このように、OCXを用いて個別に開発されたアプリケーションプログラムの汎用性を高めることも可能である。

【0039】第2のレベルのOCXとしてレシートプリンタ制御用OCX75、スリッププリンタ制御用OCX76、キャッシュドロワ制御用OCX77およびカスタマディスプレイ制御用OCX78が用いられている。これらのOCX75～78は、アプリケーションプログラムあるいは上位のOCXに対し、所定の仕様のインタフェース（API）を提供するOCXである。従って、POSシステムを構成するプリンタ等の周辺装置のメーカーや機種に係わりなく、アプリケーションプログラムや上位のOCXは所定の仕様でデータを提供すれば良い。このレベルのOCX75～78は、周辺装置固有の仕様が反映された下位のドライバレベルのOCXのプロパティを得て、共通の仕様で入力されたデータを下位のOCX、すなわち、実際にシステムを構築している周辺装置の仕様に合わせたデータに変換する。

【0040】第3のレベルのOCXは、プリンタドライバOCX91および92と、キャッシュドロワドライバOCX93、およびディスプレイドライバOCX94である。これらのOCX91～94は、個々の周辺装置に対応したコモンオブジェクトのインスタンスであり、通常はメーカー毎あるいは機種毎に異なり周辺装置と共に提供されるものである。これらのドライバレベルのOCX91～94は、例えば、最大印字行数や印字行ピッチと言ったプリンタ固有の仕様や設定状態（プリンタステータスと称する）をプロパティとして備えており、これらのプロパティは上位のOCXやアプリケーションプログラムで参照できるようになっている。また、指定された位置に文字列を印字する、すなわち、印字命令を出力するというメソッドや、プリンタステータス等のプロパティなどを備えている。従って、ドライバレベルである第3レベルのOCXからは、例えば、プリンタ用のOCXであれば、送られた印字位置および印字文字列のデータに基づき、改行量（すなわち行ピッチ）、改行コマンド、印字データおよび印字コマンド、オートカットコ

マンドが所定の順序でポートドライバ100を介してプリンタに送信される。

【0041】第3レベルのOCXは、プリンタから送信される処理結果およびエラーステータス等の非同期に発生するアクションも受け取る。そして、これらのアクションの内容を対応する周辺装置の仕様にに基づき解釈し、普遍的な形式に変換してイベントとして上位のOCXに返す。本例の制御システムでは、双方向の通信が可能なインタフェースオブジェクト81a、81b、82a～82dを用いて上位のOCXと下位のOCXを接続している。プロパティやメソッドに加え、イベントも伝達される。従って、第2レベルの制御用OCX75～78はイベントを受けて、そのまま、あるいはさらに普遍的な形式に変換し、インタフェースオブジェクト82a～82dを介して上位のOCXやアプリケーションプログラムに伝達する。これに基づき、アプリケーションシステムや上位のOCXは、ユーザーに対してメッセージを出力したりエラー処理ルーチンを起動するなどの処理を行えるので、本例の制御システムにおいては、非同期に発生するアクションに対して迅速にそして的確な処理を行うことができる。

【0042】このように、本例の制御システムは、OCXを用いているのでユーザーの採用する周辺装置に適合したカスタマイズが容易に行えるシステムであり、また、OCX間でプロパティ、メソッドおよびイベントの双方向の通信が確保されているので、処理速度が早く的確に処理が行える制御システムである。

【0043】レシート印字機能を例にとりてさらに詳しく説明する。第3レベルのOCXであるプリンタドライバOCX91は、レシートプリンタ112のプリンタステータスをプロパティとして備えている。また、プリンタ112に印字命令を出力するメソッドを備えている。さらに、プリンタ112からリアルタイムで送られてくる用紙なし（ランアウト）やカバーオープンといったエラーステータスに対応して非同期にイベントを発生する。

【0044】アプリケーションプログラム71からデータが第1レベルのレシートプリンタフォーマット変換用OCX73に渡され、所定のフォーマットに変換されたのち、第2レベルの制御用OCX74に渡される。そして、制御用OCX74がドライバOCX91の印刷を実行するメソッドをコールすると、ドライバOCX91がレシートプリンタ112のプリンタステータスに合致した適当なコマンドおよびデータをレシートプリンタ112に対し送信し、レシート印字を実行する。レシートプリンタ112は、送信されたコマンドを実行し、所定のステータスをドライバOCX91に返す。ドライバOCX91は、返されたステータスを解釈し、エラーステータスがアクティブでなければイベントを発生せず、印刷処理は終了する。また、上位のOCXであるレシートブ

リント制御用OCX75は、プリンタステータスのプロパティを参照することによってプリンタ処理の結果を知ることもちろん可能である。これらの通信はインタフェースオブジェクト82aを介して行われる。

【0045】一方、印刷実行のコマンドを実行した結果、あるいは待機中にレシートプリンタ112にエラーが発生してエラーステータスがアクティブになると、プリンタ112はエラーステータスをドライバOCX91に送信する。この場合、ドライバOCX91はイベントを発生し、エラーの発生を上位の制御用OCX74にインタフェースオブジェクト82aを介して伝達する。制御用OCX74は、イベントに対応して特定の処理を行うことも可能であるし、さらにインタフェースオブジェクト81aを介してフォーマット変換用OCX74およびアプリケーションプログラム71に通知し、注意を喚起することによって所定の処理を行わせることも可能である。

【0046】これらのプロパティ、メソッドおよびイベントに加え、スリップリンタ113に対応したドライバOCX92としては、スリッ印字に特有のプロパティおよびイベントを設けておくことが望ましい。例えば、プロパティにスリッ用紙の有無を加えることによって、上位のOCXやアプリケーションが印刷実行を指示するタイミングの判断が可能となる。また、イベントに用紙挿入検出、用紙終端検出を加えることにより、アプリケーションプログラムなどにスリッ印刷後の処理やエラー処理をスムーズに行わせることができる。

【0047】キャッシュドロー制御用OCX77はキャッシュドロードライバOCX93と通信する。キャッシュドロードライバOCX93はキャッシュドローオープンメソッドとして備えており、キャッシュドローのオープン・クローズを含めたキャッシュドローステータスをプロパティとして備えている。また、イベントとしてキャッシュドローエラーやキャッシュドローオープン検出を発生することができる。

【0048】カスタマディスプレイ制御用OCX78は、ディスプレイドライバOCX94と通信し、このドライバOCX94は、表示位置や表示文字列を指定した表示命令をメソッドとして備えている。また、プロパティとしては表示桁数や表示色などのディスプレイ仕様を備えており、ディスプレイ制御用OCX78はこのプロパティに沿って表示データを与える。また、カスタマディスプレイはイベントに該当するエラー等の発生は少ないので、イベントを発生する機能を削除することも可能である。

【0049】このように、本発明に係るインタフェースオブジェクトを採用することにより、複数のOCXを用いて階層的な構造を備え、エラー等を含めた非同期に発生するアクションに対しても迅速に対処可能な制御システムを構築できる。また、本例の制御システムは、パソ

コンに接続される周辺装置に対しフレキシブルに対応できるオープンな制御システムである。例えば、プリンタ 111 は、レシート印字機能 112 と、スリップ印字機能 113 と、さらにドロワ 115 の制御機能とを備えており、制御システムはそれぞれの機能毎に OCX 91~93 が用意されている。このプリンタ 111 に対し、スリップ印字機能の仕様（例えば、最大印字桁数、実行可能なコマンドセット等）の異なるプリンタに置き換える場合は、スリップ印字機能を制御するドライバ用のコモンオブジェクトを新しいプリンタの仕様に対応するものに置き換えれば良い。周辺装置の制御システムにおいては、置き換えられた新しいドライバ用のコモンオブジェクトから OCX を作成し新しいプリンタに適合した制御システムを自動的に形成する。プリンタがバージョンアップされて仕様が変った場合でも、同様にドライバ用のコモンオブジェクトを置き換えるだけで良く、その他のコモンオブジェクトやアプリケーションプログラムに変更を加える必要はない。

【0050】カスタマディスプレイについても同様であり、仕様が変更された場合等はドライバ用のコモンオブジェクトを変更すれば良い。また、バーコードリーダ等の他の周辺装置が加えて POS システムが構成される場合は、それに対応した制御用のコモンオブジェクトとドライバ用のコモンオブジェクトがパソコンのシステム内に用意されておれば、アプリケーションあるいは上記の OCX によって、下位の OCX が作成され、バーコードリーダ等を含めた周辺装置の制御システムが形成される。

【0051】同様に、アプリケーションプログラムとのインタフェースを担当する上位の OCX がパソコンのハード等に起因する仕様の違いを吸収し、アプリケーションに対し共通のインタフェースを与えるものであれば、パソコンのハード等の相違とは無関係に共通のアプリケーションプログラムを用いてシステムを構築することも可能となる。パソコン内に用意されたコモンオブジェクトから制御システムがアプリケーションプログラムによって作成され、制御システムを構成する OCX によってパソコンおよびそれに接続された周辺装置に適合したシステムが自動的に形成される。このように、本発明の OCX を用いた制御システムはオープンなシステムであり、インタフェースの仕様を予め合意しておけば、アプリケーションを供給する側は、パソコンやそれに接続される周辺機器の仕様および接続方法等は一切考慮せずにアプリケーションを開発し供給することができる。従って、短期間で開発が可能となり、安価に提供できる。また、ユーザーもパソコンや周辺装置に係わりなく、自己の目的や環境に適したアプリケーションを自由に選択することができる。

【0052】一方、周辺装置を提供する側も、供給する周辺装置の仕様に対応したコモンオブジェクトを用意

し、例えば、周辺装置と共に供給することによって、パソコンやアプリケーションに限定されない汎用的な周辺装置を提供することができる。従って、ユーザーも自己の目的や環境に適した周辺装置を自由に購入し、システムを構築することができる。

【0053】さらに、本発明に係るインタフェースオブジェクトを採用することによって、コモンオブジェクトおよびそのインスタンス同士の間で双方向にプロパティ、メソッドおよびイベントの伝達が可能となる。従って、本発明の基づく制御システムでは、システムの応答が遅れたり、システムの機能が限定される等のシステムをオープン化する上での弊害は全くない。また、制御システムの機能アップやバージョンアップも個々の制御オブジェクト毎に行うことができるので、システム開発を効率良く、短期間に行える。また、ユーザーもシステムのグレードアップや変更等を簡単にできるようになる。

【0054】なお、上記では、用いられる周辺装置の種類や数が多く、また、ユーザーの環境に合わせて様々な機種の周辺装置が用いられる例として POS システムを説明しているが、POS システムに限定されないことはもちろんである。近年のパソコンを中心としたシステムは、ユーザーの目的や能力に合わせて様々な仕様の周辺装置が組み合わされるようになりつつある。本発明に係る共通する制御オブジェクトを用いた制御システムは、様々なユーザーの要望に対してフレキシブルに対応可能なシステムであり、POS システムに限らず、今後様々なシステムに対して適用可能である。

【0055】

【発明の効果】以上に説明したように、本発明においては、アプリケーションプログラム間で共通に活用できる制御オブジェクトであるコモンオブジェクトおよびそのインスタンスを用いて制御システムを構築している。さらに、インタフェースオブジェクトによって、これらコモンオブジェクトおよびそのインスタンス同士の間で双方向の通信を可能としている。従って、コモンオブジェクトを活用して多種多様なオペレーティングシステムあるいはアプリケーションプログラムを簡単に構築し、提供することができる。このように本発明により、入出力端末や、処理する対象などにより多種多様に変化するコンピュータを中心とした近年のシステムにおいて、いっそうフレキシブルに、そしてカスタマイズ可能なシステムを実現するのに好適な制御システムおよびその構築方法を提供できる。

【0056】特に、上述したようなコンピュータの機種やそれに接続される周辺装置の種類および数量がユーザーの環境に応じて多種多様に変化する制御システムを構築するのに本発明は好適である。本発明に係る制御システムを用いることにより、アプリケーションプログラムに対し周辺装置に依存しない共通のアプリケーションプログラムインタフェースを提供可能である。さらに、

本発明に係る制御システムは、汎用性の非常に高い制御システムでありながらエラー等の非同期に発生するアクションに対しても迅速にアプリケーションを作動させることが可能である。従って、ユーザーに対し自己の目的に対応した快適な動作環境を容易に構築できる制御システムを提供できる。

【図面の簡単な説明】

【図1】本発明の実施例の制御システムの概略構成を示すブロック図である。

【図2】図1に示す制御システムにおいて、インタフェースオブジェクトとオブジェクトインスタンスとの間でコネクションを確認する過程を示すフローチャートである。

【図3】図1に示す制御システムにおいて、インタフェースオブジェクトとオブジェクトインスタンスとの間でイベントの対応付けを行い、コネクションを張る過程を示すフローチャートである。

【図4】図1に示す制御システムにおいて、オブジェクトインスタンス側からイベントをインタフェースオブジェクトに伝える過程を示すフローチャートである。

【図5】上記と異なる本発明の実施例の制御システムの概略構成を示すブロック図である。

【図6】本発明の制御システムによって構築されたPOSシステムの周辺装置を制御するシステムを示すブロック図である。

【符号の説明】

10・・・第1のOCX

11・・・第2のOCX

15・・・インタフェースオブジェクト

20・・・アプリケーション

70・・・パソコン

71・・・POSアプリケーションプログラム

72・・・制御オブジェクトによって構成された制御システム

100・・・ポートドライバ

110・・・カスタマディスプレイ

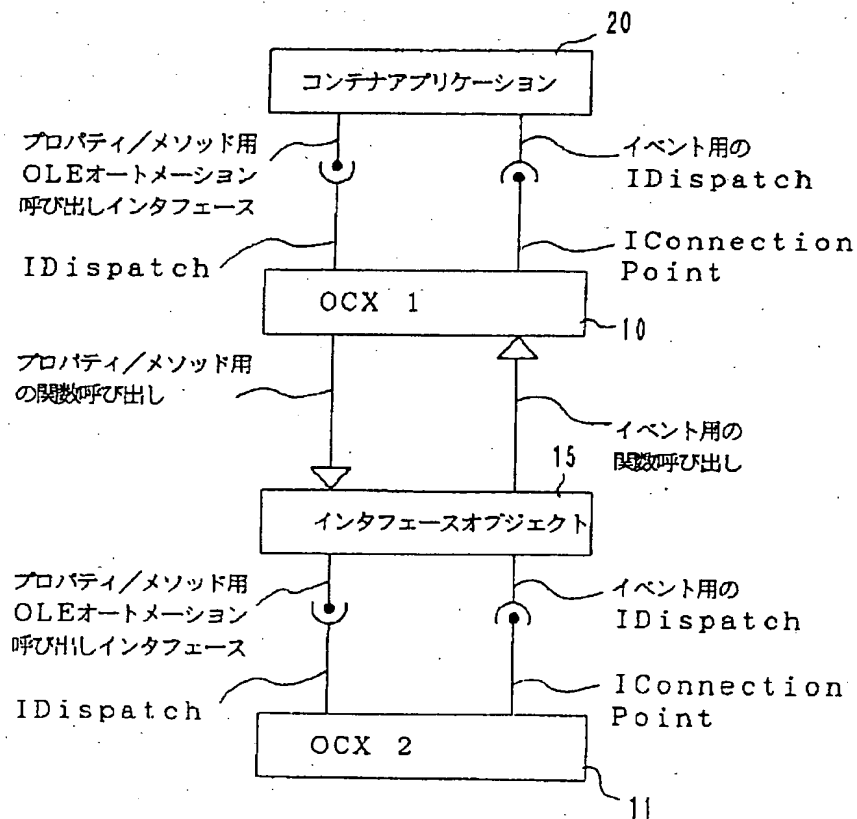
111・・・プリンタ

112・・・レシート印字機構

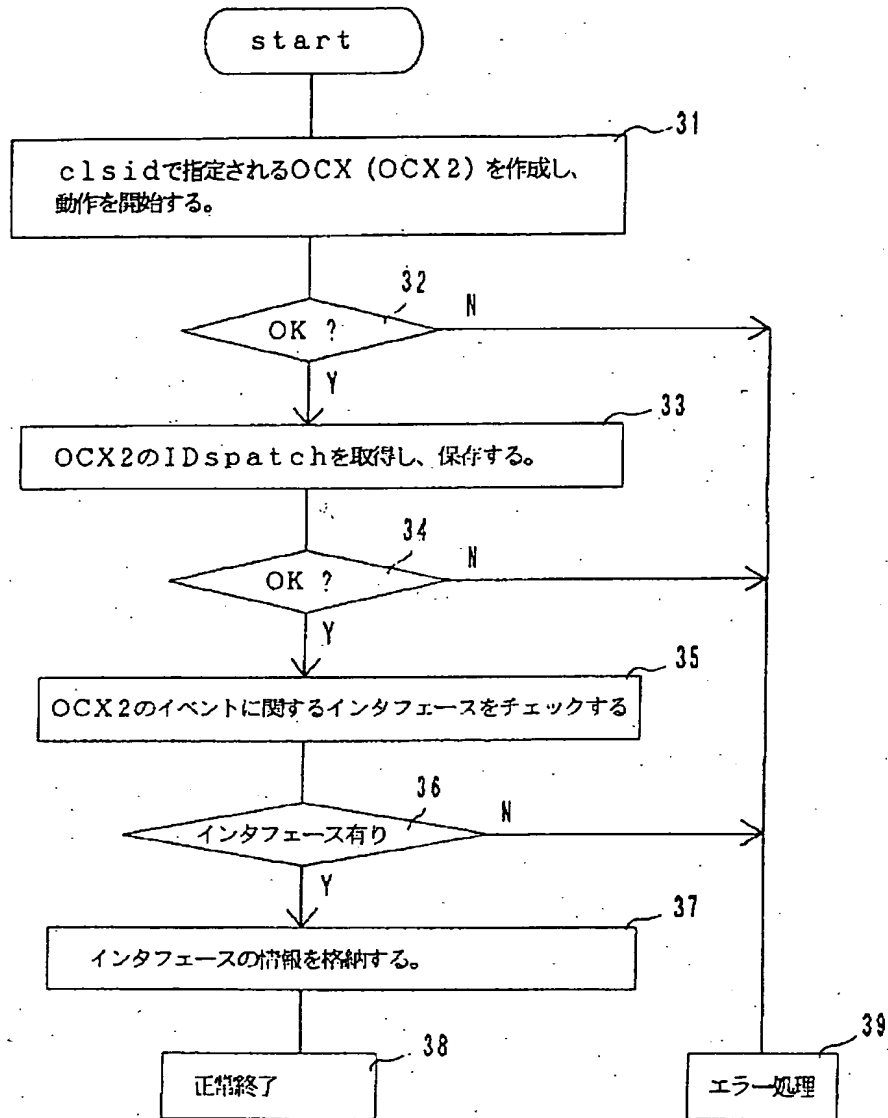
113・・・スリップ印字機構

20 115・・・キャッシュドロワ

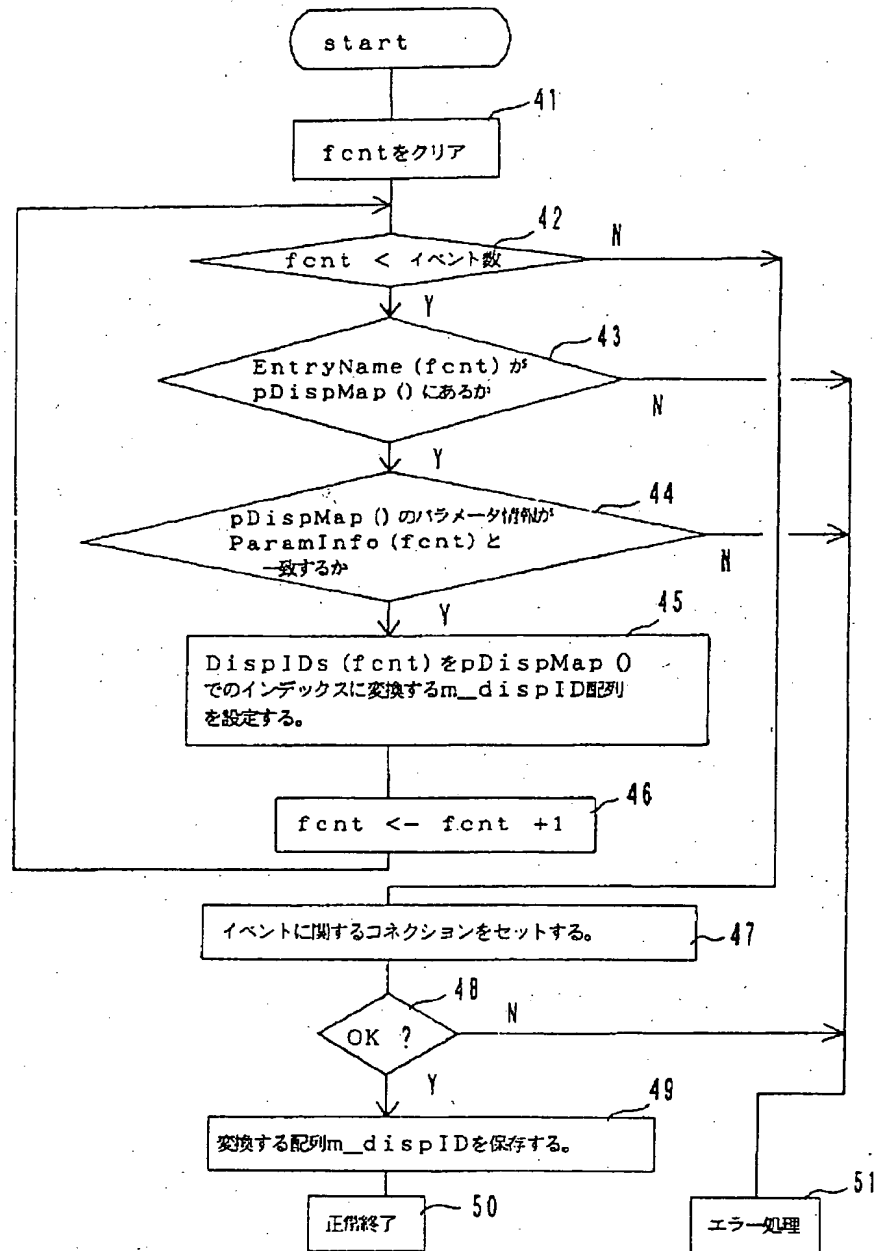
【図1】



【図2】

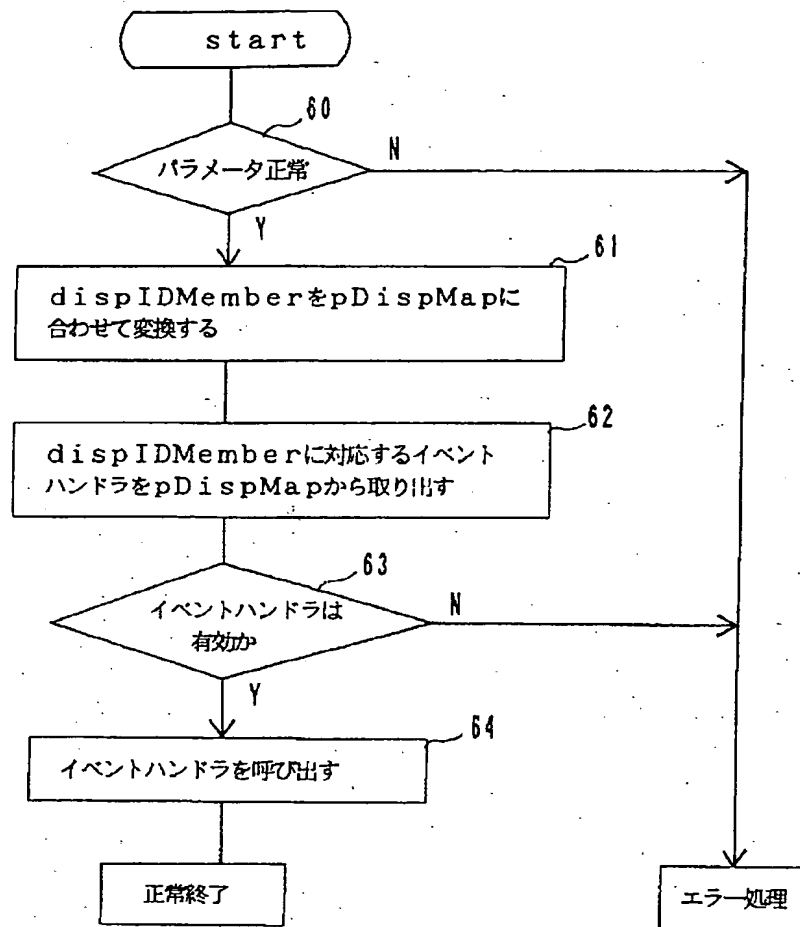


【図 3】

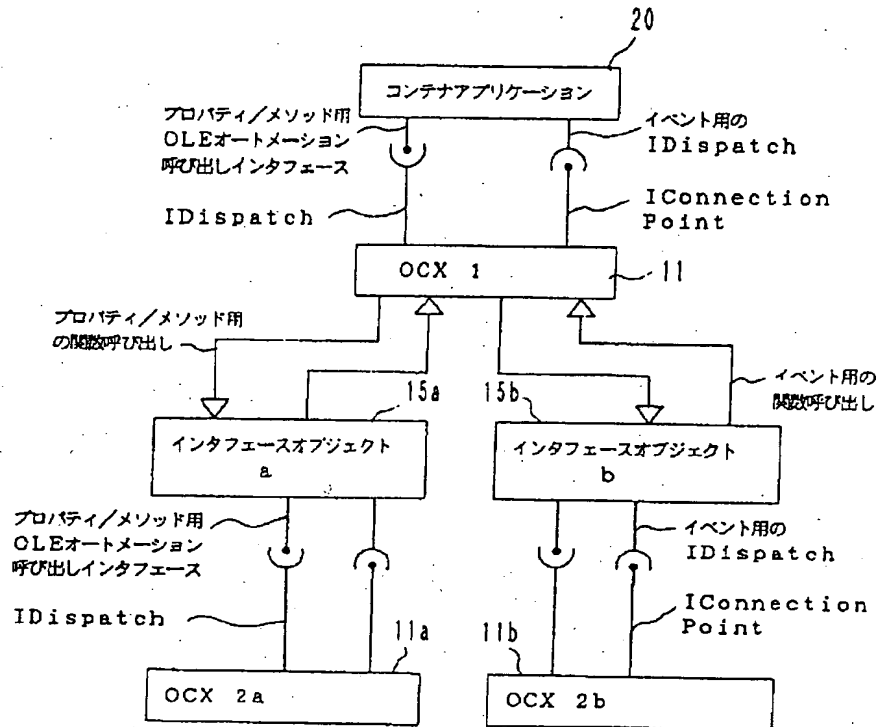




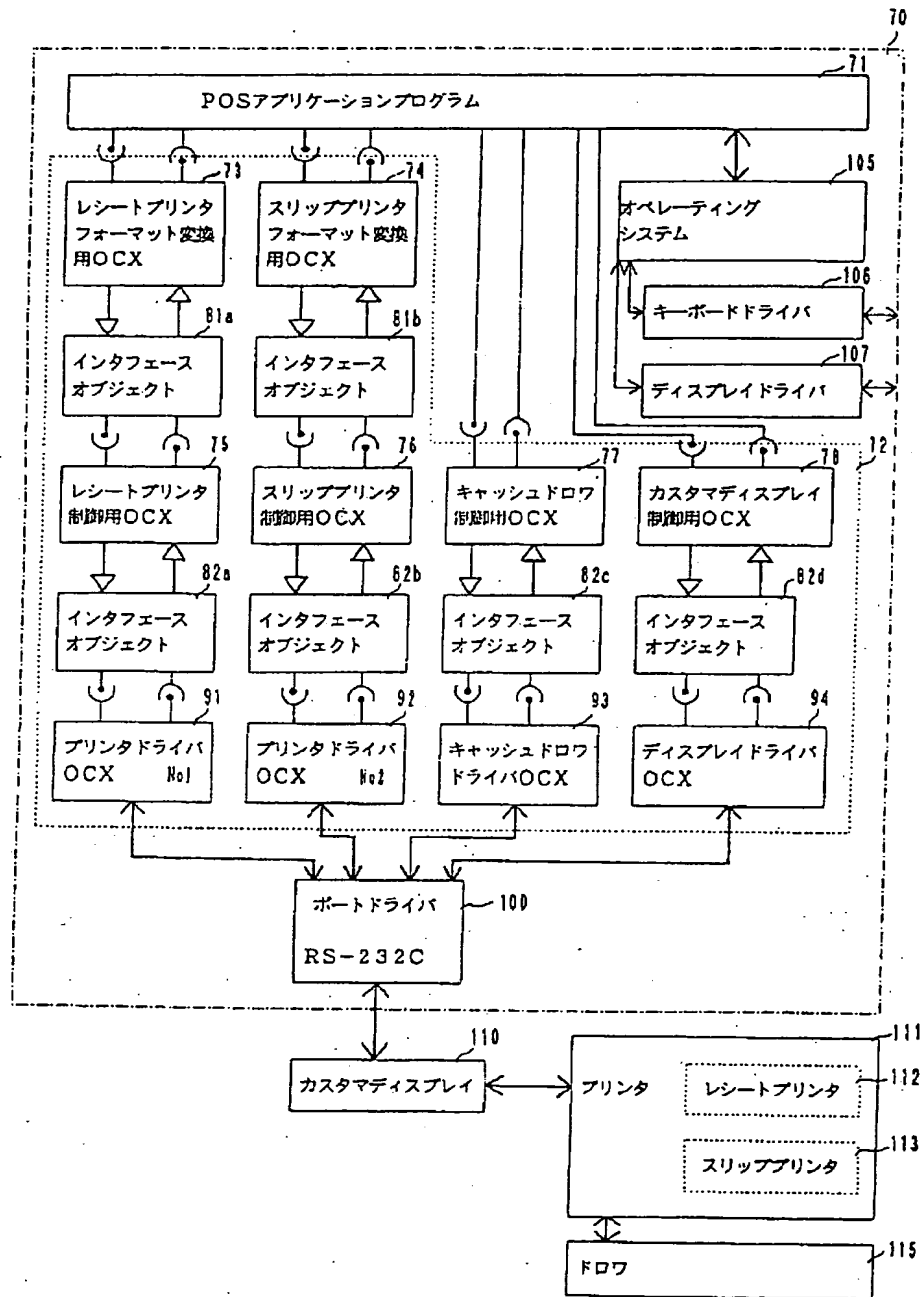
【図4】



【図5】



【図 6】



THIS PAGE BLANK (USPTO)